Operating systems

MP1

Team 67

108062128林育丞

108062135呂佳恩

|  |  |
| --- | --- |
| 組員 | 內容 |
| 林育丞 | 報告製作與實作Part II - 2 |
| 呂佳恩 | Trace Code與實作Part II - 2 |

分工表

(a)、(b) Trace Code, explanations of system calls

1. SC\_Halt
2. **mipssim.cc**  
    Machine::Run( )   
   在UserMode下，不斷地執行以下兩個行為:  
    OneInstruction() : 處理User-level program  
    OneTick() : 推進模擬的時間，或是處理可能到來Pending Interrupt。  
    Machine::OneInstruction( )   
   1. 讀取registers[PCreg]存的資料並解析成Opcode。  
   2. 接著以龐大的switch-case架構，將instr->Opcode做解析，並執行對應的行為

在本題的情況下，instr->Opcode為OP\_SYSCALL，其所對應的行 為是呼叫RaiseException()，並傳入SyscallException。

1. **machine.cc**  
    Machine::RaiseException(ExceptionType which, int badVAddr)   
   在執行User-level program的時候碰到錯誤(或是例外情形)，就會呼叫這個函式。此時會記錄出錯的virtual address(參數badVAddr)，接著將模式改為SystemMode (也就是Kernel Mode)，之後將錯誤名稱的參數which傳給ExceptionHandler。  
   在ExceptionHandler()執行完畢之後，會切換回UserMode。

在本題的情況下，錯誤名稱的參數內容即是SyscallException，而出錯的address為0 (因為這個Exception是System Call，並非錯誤，所以為0)。

1. **exeception.cc**  
    ExceptionHanler(ExceptionType which)   
   解析傳入的Exception Type，如果是SyscallException，則根據register[2]內容的Type，去執行相應的處理。

在本題的情況下，Type為SC\_Halt，而相應的處理則是呼叫SysHalt()。

1. **ksyscall.h**  
    SysHalt( )   
   SysHalt( )是作為kernel Interface for system calls

定義遇到不同systemcall 該找誰處理

在本題的情況下，是呼叫kernel->interrupt->Halt()

1. **interrupt.cc**  
    Interrupt::Halt( )   
   呼叫kernel->stats->Print()，輸出當前資訊(如totalTicks)，並刪除kernel (停止這個kernel的運作)。
2. SC\_Create
3. **exception.cc**  
    ExceptionHandler(ExceptionType which)   
   解析傳入的Exception Type，如果是SyscallException，則根據register[2]的內容(Type)，去執行相應的處理。
4. 在本題的情況下，Type為SC\_Create，而相應的處理如下 :   
    1. 首先讀取register[4]的內容(檔案名稱在主記憶體中的位置)，   
    2. 接著呼叫SysCreate(filename)，並將執行結果存回register[2]，   
    3. 最後推進PC counter，以免無限地重複執行這個Exception。
5. **ksyscall.h**  
    SysCreate(char \*filename)   
   kernel Interface for system calls  
     
   在本題的情況下，是呼叫kernel->fileSystem->Create(filename)，並回傳成功與否。
6. **filesys.h**  
    FileSystem::Create(char \*name, int initialSize)   
   使用非Stub的File System。

首先建立一個Directory，並從根目錄開始，去查詢欲新增的檔案名稱 (參數name)是否已經存在，如果存在就回傳FALSE。   
否則，會在判斷是否有足夠儲存空間之後，取得一塊參數initialSize大小的空間，然後建立一個新的檔案，並回傳TRUE。

1. SC\_PrintInt
2. **exception.cc**  
    ExceptionHandler(ExceptionType which)   
   解析傳入的Exception Type，如果是SyscallException，則根據register[2]的內容(Type)，去執行相應的處理。  
     
   在本題的情況下，Type為SC\_PrintInt，而相應的處理如下 :   
   首先讀取register[4]的內容(欲輸出的數字)，   
   接著呼叫SysPrintInt(val)，   
   最後推進PC counter，以免無限地重複執行這個Exception。
3. **ksyscall.h**  
    SysPrintInt(int val)   
   kernel Interface for system calls  
     
   在本題的情況下，是呼叫kernel->synchConsoleOut->PutInt(val)
4. **synchconsole.cc** SynchConsoleOutput::PutInt(int value)   
   先以sprintf將欲參數value格式化，並存入str變數中。  
   使用lock->Acquire()，使得只有當前這個Thread可以執行Output動作。  
   然後不斷執行PutChar，將str裡面的字元一個一個傳入，並且等待P()，  
   直到當前字元為結束字元’\0’為止。  
   使用lock->Release()，讓Output可再次被其他Thread使用。  
     
    SynchConsoleOutput::PutChar(char ch)   
   執行lock->Acquire()。  
   執行consoleOutput->PutChar()，傳入剛剛傳入的字元，並且執行waitFor->P( )，等待CallBack function呼叫waitFor->V()。  
   執行lock->Release()。
5. **console.cc** ConsoleOutput::PutChar(char ch)   
   呼叫WriteFile()，將參數ch放到輸出模擬之中。  
   並以putBusy來確保同時只會有一個字元被輸出。  
   Schedule一個100 ticks以後的Interrupt來做為讀寫一個字元的耗時。  
     
   在本題的情況下，這個Thread會進入睡眠模式。  
   如果沒有其他Thread要執行，則Interrupt會進入Idle狀態，並跳至下一個Interrupt到來的時間點(也就是100 ticks之後)。
6. **interrupt.cc**  
    Interrupt::Schedule(CallBackObj \*toCall, int fromNow, IntType type)   
   用上一個函式傳來的所有資訊，建立PendingInterrupt的Instance，並將這個Instance排入pending List之中，等待其執行的時機到來。
7. **mipssim.cc**  
    Machine::Run()   
   執行OneInstruction來處理其他可以執行的指令，和使用OneTick來推進時間。
8. **interrupt.cc**  
    Machine::OneTick( )   
   依據現在的是SystemMode或是UserMode，去增加各自的Tick數。  
   並且在關閉Interrupt之後，使用CheckIfDue()檢查是否有即將到來的Interrupt。  
     
    Interrupt::CheckIfDue(bool advancedClock)   
   檢查是否有已經到來的Pending Interrupt，如果有，則執行其CallBack函數。  
   如果當前的Interrupt是處於Idle狀態，則參數advanceClock會是TRUE，  
   此時如果有尚未到來的Pending Interrupt等待執行，則會直接將Ticks跳至該Pending Interrupt的時間。
9. **console.cc**  
    ConsoleOutput::CallBack( )   
   Pending Interrupt到來後，會執行這個CallBack函式。  
   將putBusy設為FALSE，並記錄已經寫入幾個字元。  
   呼叫下一層的CallBack。  
     
   在本題的情況下，之前預約的Pending Interrupt預計會在100 ticks之後到來。
10. **synchconsole.cc**  
     SynchConsoleOutput::CallBack( )   
    呼叫waitFor->V( )，以解除之前Thread的睡眠狀態。  
      
    在本題的情況下，這個Thread會繼續執行consoleOutput->PutChar()，並且再次睡眠，然後再次等待解除睡眠，直到輸出完畢。

(c) Implementation

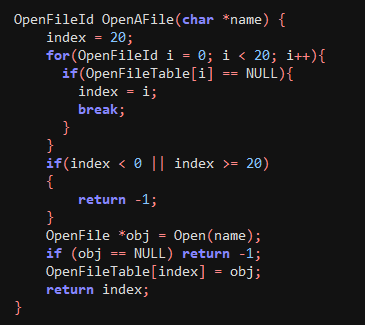
我們主要的想法，是先將能用的資源一步步取得，再去寫底層實作的部分。

1. 在syscall.h中，將定義好的SC\_Open、SC\_Read等取消註解。
2. 在exception.cc中，依據其他已經寫好的Case，來新增我們需要的四種Case，並依據所需的參數，去存取registers的值。
3. 在ksycall.h中，模仿其他已經寫好的Interface，將四個函式介面和其所需的參數寫好。
4. 接著在Start.S中，依據其他Exception的寫法，新增四個Exception。

完成了簡單的部分，接著就是實作函式內容。大部分的改動都在filesys.h裡面。而因為我們被要求使用Stub File System，所以不需要去改動filesys.cc。

1. OpenAFile( char \*name)

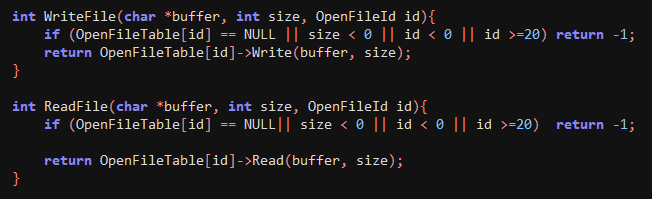
這部分本來spec 的第二點有規定不可使用sysdep.h當中的函數來access file, 因此我們使用 Open( )來時做這部分，不過OpenFile 回傳的資料，在id方面是一個private 的變數，因此無法讀取他，這是我們遇到的最大困難。解決辦法最後是使用一個for迴圈去尋找OpenFileTable當中目前空下的欄位。



初始化將index設為20，這樣未尋找到時就會進入下方判斷式中，return -1，同時我們也需要判斷找到的位置是否超過開啟的限制，最後將改動OpenFileTable.

1. WriteFile, ReadFile

這兩個函數實作方面比較相似



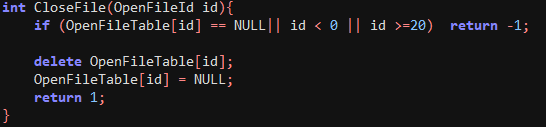
在判斷完id 是否為valid後，並且判斷size為正常的輸入，呼叫相對應函數

1. CloseFile

首先判斷 id 是否為valid

在用 destructor 去刪除OpenFile object

最後將OpenFileTable欄位清空



(d) What difficulties did you encounter when implementing this assignment?

Tracing the code at first was quite complicated, since there are multiple files linking to each other. However, after spending a lot of time, we slowly began to understand how the system worked.

Implementing was hard to begin, due to us not knowing where to start. We then opened up the fileIO\_test1 and understand how the functions were being called at the begging, we then followed the steps and the understanding we had from tracing the code.

Initially, the second point of the spec requiring us not using functions from the sysdep.h file made it hard for us to implement Open. However, we overcame the obstacle.

(e) Any feedback you would like to let us know.

It would be nice to know earlier that we could use functions from sysdep.h.

We spend multiple hours trying to solve the problem. After solving it, we then received an email saying that we could use the functions. It would be nice to know that we could use the functions in the first place.

We learned a lot from this project and have a deeper understanding of how the system calls work!